САБИТОВ Б.Р., ДЖЕНТАЕВА К.Б., КАЛИЛ УУЛУ Б.
Кыргызский Национальный Университет им.Ж.Баласагына
УДК.004.42

СОЗДАНИЕ СОБСТВЕННОГО ПРОИЗВОДИТЕЛЬНОГО ФРЕЙМВОРКА PHP НА ОСНОВЕ АРХИТЕКТУРЫ МПК.

Аргументирована эффективность и легкость написания собственного фреймворка для построения небольших веб-сайтов.

Өтө чоң эмес сайттарды өз колу менен жазылган фреймворк менен түзүү натыйжалуу экени тастыкталган.

Is given reason efficiency and ease writing custom framework for building small websites.

На сегодняшний день, скриптовый язык PHP, является самым популярным языком в области построения веб-сайтов, PHP завоевывала популярность за счет по своей простоте и появлением разных типов Фреймворков к нему. Однако в этих фреймворках есть много практически не нужных и не используемых функционалов для построения небольших проектов. Если исходить по принципу Парето 20/80 «20 % функционала дают 80 % результата, а остальные 80 % функционала — лишь 20 % результата», В этой статье попытаемся реализовать фреймворк стандартными средствами PHP и шаблонами проектирования, обхватывающий 20 % функционала для построения веб-сайтов 80 % продуктивности.

Модель-Представление-Контроллер — наиболее известный принцип архитектуры программного обеспечения, в которой модель данных приложения, пользовательский

интерфейс и управляющая логика разделены на три отдельных компонента, так, что модификация одного из компонентов оказывает минимальное воздействие на другие компоненты.Ниже рассмотрим концептуальную схему шаблона МПК:

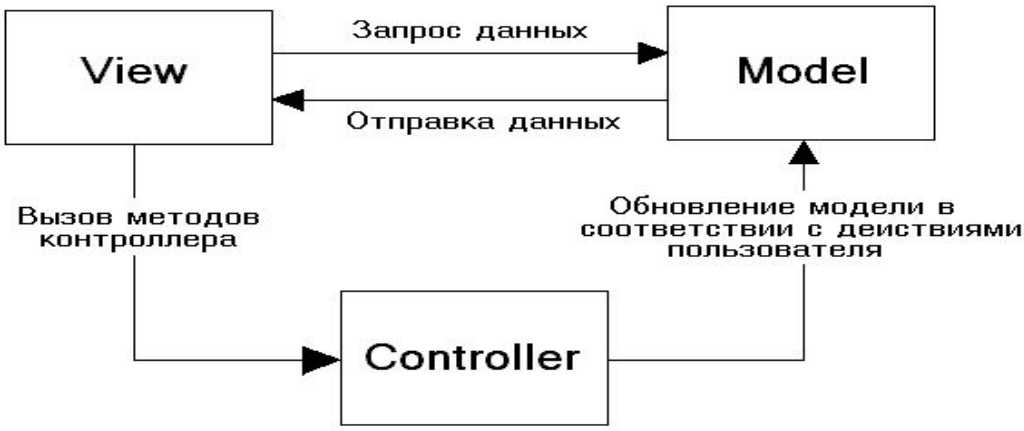

Рисунок 1

В архитектуре МПК модель предоставляет данные и правила бизнес-логики, представление отвечает за пользовательский интерфейс, а контроллер обеспечивает взаимодействие между моделью и представлением.

Алгоритм работы МПК можно охарактеризовать следующим образом:

1. При заходе пользователя на веб-ресурс, скрипт инициализации создает экземпляр приложения и запускает его на выполнение.
2. Приложение получает запрос от пользователя и определяет запрошенные контроллер и действие. В случае главной страницы, выполняется действие по умолчанию.

3. Приложение создает экземпляр контроллера и запускает метод действия, в котором, к примеру, содержаться вызовы модели, считывающие информацию из базы данных.

4. После этого, действие формирует представление с данными, полученными из модели и выводит результат пользователю.

Модель — содержит бизнес-логику приложения и включает методы главные задачи модели заключаются в осуществлении доступа к данным для их просмотра или актуализации (добавления, редактирования, удаления). Модели являются связующим звеном между представлениями и контроллерами. Модель не должна напрямую взаимодействовать с пользователем. [2, С250]

Представление — используется для задания внешнего вида отображения данных, полученных из контроллера и модели. Виды содержат HTML-код и небольшие вставки PHP-кода для обхода, форматирования и отображения данных, где осуществляется вывод данных на экран. [2, С250]

Контроллер — связующее звено, соединяющее модели, виды и другие компоненты в рабочее приложение. Контроллер отвечает за обработку запросов пользователя. Контроллер не должен содержать SQL-запросов. Их лучше держать в моделях. Контроллер не должен содержать HTML-код и другой разметки. Её стоит выносить в виды. После того как контроллер заполучил данные, в зависимости от нужной задачи, он передаст данные в представление для вывода или в модель для актуализации (добавления, редактирования, удаления). [2, С250]

Front Controller - является диспетчером запросов и, в зависимости от URL запускает нужный контроллер с нужными параметрами. В этом ему помогает Router, занимающийся непосредственно разбором URL и применением различных правил. Это позволяет сократить дублирование и уменьшить вероятность ошибок.

Реализация маршрутизатора URL с помощью .htaccess:

```
RewriteEngine On //Включение модуля ModRewrite
RewriteCond %{REQUEST_FILENAME} !-f //Опередение параметров запроса
RewriteCond %{REQUEST_FILENAME} !-d//Опередение параметров запроса
RewriteRule .* index.php [L]//Определение правил преобразований
```

Структура файлов представлена на рисунке 2

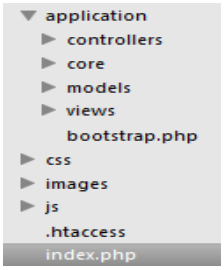

Рисунок 2

Файл index.php содержит следующий код:

```
require_once 'application/bootstrap.php'; //Подключаем все нужные классы
```

Файл bootstrap.php содержит следующий код:

```
require_once 'core/model.php';//Подключение файла ядра Модели
require_once 'core/view.php';  // Подключение файла ядра Представления
require_once 'core/controller.php'; // Подключение файла ядра Контроллера
require_once 'core/route.php'; //Подключение файла маршрутизатора
Route::start(); // Запуск маршрутизатора
```

Реализация маршрутизатора route.php:

```
class Route{
    private static $_instance=null;
    private $modelName, $modelFile, $modelPath, $controllerName="Main", $controllerFile, $controllerPath, $actionName="index", $parameterKey;
    private $routes;
    private function __construct() {}
    private function __clone(){}
    public static function getInstance()
    {
    if (self::$_instance == null){
    self::$_instance = new self;
     }
     return self::$_instance;
    }
    public function run()
    {
        $this->routes = array();
        $this->routes = explode("/", $_SERVER['REQUEST_URI']);
        reset($this->routes);
        $this->controllerName = !empty($this->routes[2])?$this->routes[2]:$this->controllerName;
        $this->actionName = !empty($this->routes[3])?$this->routes[3]:$this->actionName;
        $this->parameterKey = !empty($this->routes[4])?$this->routes[4]:$this->parameterKey;
        $this->modelName = "Model_".$this->controllerName;
        $this->controllerName = "Controller_".$this->controllerName;
        $this->actionName = "action_".$this->actionName;
        $this->modelFile = strtolower($this->modelName).".php";
        $this->modelPath = "application".DIRECTORY_SEPARATOR."models".DIRECTORY_SEPARATOR.$this->modelFile;
        if (file_exists($this->modelPath)){
            include($this->modelPath);
        }
        $this->controllerFile = strtolower($this->controllerName).".php";
        $this->controllerPath = "application".DIRECTORY_SEPARATOR."controllers".DIRECTORY_SEPARATOR.$this->controllerFile;
        try{
            if (file_exists($this->controllerPath)){
                include($this->controllerPath);
            }else{
                throw new ApplicationException("Извините такой страницы не существует");
            }
        } catch (ApplicationException $exp){
            die($exp->getMessage());
        }
        try{
```

```
            $objectConroller = new $this->controllerName();
            $method        = $this->actionName;
            if (method_exists($objectConroller, $method)){ empty($this-
>parameterKey)?$objectConroller->$method():$objectConroller->$method($this-
>parameterKey);
            }
            else{
               throw new ApplicationException("Такого метода не существует");
            }
         } catch (ApplicationException $exp){
            die($exp->getMessage());
         }
      }
   }
```

Содержимое файла model.php:

```
class Model
{
      public  function get_data()
   {
      }
}
```

Класс модели содержит единственный пустой метод выборки данных, который будет перекрываться в классах потомках.

Содержимое файла view.php

```
class View
{
      function generate($content_view, $template_view, $data = null)
   {
            include  'application/views/'.$template_view;
      }
}
```

Не трудно догадаться, что метод *generate* предназначен для формирования вида. В него передаются следующие параметры:

1. *$content_file* — виды отображающие контент страниц;
2. *$template_file* — общий для всех страниц шаблон;
3. *$data — массив*, содержащий элементы контента страницы. Обычно заполняется в модели.

Функцией *include* динамически подключается общий шаблон (представление), внутри которого будет встраиваться вид для отображения контента конкретной страницы.

Содержимое файла controller.php:

```
class Controller {
      public  $model;
      public  $view;
      function __construct()
   {
            $this->view = new View();
      }
      function action_index()
   {
      }
}
```

Метод *action_index* — это действие, вызываемое по умолчанию.

Файл главной страницы контроллера controller_main.php:

```
class Controller_Main extends Controller
{
    function action_index()
  {
        $this->view->generate('main_view.php', 'template_view.php');
    }
}
```

Файл главной страницы контента main_view.php выглядит следующим образом:

```
<h1>Привет Мир!</h1>
```

Таким образом, данный фреймворк продемонстрировал простоту реализации небольших проектов используя только стандартные средства языка PHP, не прибегая громоздким фреймворкам. При этом достигать максимально нужного результата продуктивности создаваемого проекта.

Литература:

1. Алан Шаллоуей, Джеймс Р. Тротт. Шаблоны проектирования. Новый подход к объектно-ориентированному анализу и проектированию. — М.: «Вильямс», 2002. — С. 288. — ISBN 0-201-71594-5
2. Эд Леки-Томпсон, Алек Коув, Стивен Новицки, Хьяо Айде-Гудман
PHP 5 для профессионалов. — М.: Вильямс, 2006. — ISBN 5-8459-1066-8
3. Джейсон Ленгсторф. PHP и jQuery для профессионалов = Pro PHP and jQuery. — М.: «Вильямс», 2010. — С. 352. — ISBN 978-5-8459-1693-8